20
29

UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE CIENCIAS
MATEMATICAS

SISTEMAS DIALOGICOS Y SUS APLICACIONES A BASES DE CONOCIMIENTO

SANTIAGO NEGRETE YANKELEVICH

1 9 9 0

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

ÍNDICE.

ÍNDICE.

Introducción.

Hace más de dos años ingresé como becario, al entonces recién formado grupo de Inteligencia Artificial del Centro Científico IBM. El grupo llevaba algunos meses trabajando, pero Edgar González y Pablo Noriega ya habían publicado, para el momento del inicio del proyecto, algunos artículos que contenían los cimientos filosóficos sobre los que se construiría el trabajo de los dos años siguientes. La idea fundamental era: la aplicación de los sistemas dialógicos a la Inteligencia Artificial.

Dos artículos, escritos por Edgar y Pablo (12,13) describían los sistemas dialógicos de una manera general y bosquejaban su aplicación en la depuración de bases de datos deductivas (BDD) y sistemas basados en conocimiento (SBC).

Tal posibilidad de aplicación es particularmente clara cuando se consideran SBC que aprenden, es decir, que son capaces de incorporar, en determinado momento, nuevas reglas a su acervo de información. Estos nuevos "conocimientos" pueden hacer que la información, considerada como una teoría lógica, sea inconsistente y, por consiguiente, que del SBC se deduzcan contradicciones.

Mi primera tarea como becario dentro del grupo fué la programación de un simulador de sistemas dialógicos basado en un modelo propuesto por Paul Lorenzen (22). De este proyecto surgieron dos simuladores. El primero surgió después de algunos meses de trabajo, sólo restaba, de alguna forma probar su

eficacia. Para esto intentamos probar los axiomas de Hilbert para la lógica proposicional. Ante nuestra sorpresa, el programa no probaba todos los axiomas, algunos por pertenecer estrictamente a la lógica clásica y otros por razones que hasta entonces nos eran desconocidas.

Los problemas se aclararon cuando conseguimos un artículo en el que W.Stegmüller criticaba la adecuación del sistema de Lorenzen a la lógica intuicionista dando algunos contraejemplos (34), proponía, además, un nuevo sistema dialógico como alternativa al de Lorenzen y daba las pruebas de adecuación para éste. Este hallazgo nos devolvió la confianza en el proyecto y nos llevó a emprender la programación del sistema propuesto por Stegmüller.

Una vez terminado el segundo simulador, notamos que el costo computacional para decidir cuándo un conjunto de diálogos corresponden a una prueba dialógica de la fórmula era alto (exponencial) ya que era necesario generar el árbol de diálogos casi completo. Debido a esto, establecimos la heurística, computacionalmente más sencilla pero que abarca la mayoría de los casos de tomar el primer diálogo como representativo de los demás para determinar la validez de la fórmula en cuestión.

El siguiente paso hacia la depuración de bases de conocimiento y bases de datos deductivas fue la implementación de DETEC, un sistema para detectar y corregir inconsistencias en un conjunto de fórmulas proposicionales.

El sistema usa el demostrador de teoremas basado en el modelo de Stegmüller para detectar la inconsistencia del conjunto; para la corrección, utiliza agunas de las técnicas descritas por Hayes-Roth en (15) y propone al usuario una lista de modificaciones a las reglas que disuelven la contradicción. La decisión final sobre la modificación del conjunto de fórmulas se deja al usuario porque cuando las fórmulas representan una base de conocimiento, la elección de la modificación depende de la semántica de la base y no de la parte simplemente formal, por lo tanto un sistema puramente formal como DETEC no puede tomar tal decisión.

El sistema DETEC está hecho en VMProlog (37) y funciona para conjuntos de fórmulas proposicionales. Los conectivos lógicos se definen en prolog como operadores, y las variables proposicionales como constantes.

Cerca de la conclusión de DETEC, otra parte del grupo que había estado trabajando en sistemas expertos con el entorno para sistemas expertos de IBM, ESE (Expert System Environment) (10) sugirió que, por un lado, sería de gran utilidad someter el prototipo DETEC a bases reales representadas en ESE y, por el otro, sería muy útil contar con una herramienta para depurar bases de conocimiento en ESE, lo cual llevó a la elaboración del último sistema del proyecto: INTERESE, la interfaz entre DETEC y ESE.

Algunos resultados parciales de este proyecto se encuentran

publicados en una serie de artículos cuya referencia aparece en la bibliografía. Este trabajo de tesis contiene las propuestas teóricas de los sistemas dialógicos y la descripción de los sistemas que programé para el proyecto, mismos que constituyen una aproximación hacia el desarrollo de demostradores automáticos utilizando sistemas dialógicos. Esta tesis propone, también, que los sistemas dialógicos constituyen una herramienta para desarrollar sistemas en inteligencia artificial.

El primer capítulo describe la concepción moderna de sistema dialógico, el sistema de Lorenzen y los puntos de vista de Stegmüller con respecto al sistema de Lorenzen. El segundo, describe una formalización hecha por mí del sistema de Stegmüller, distinta a la utilizada en su artículo en las pruebas de adecuación y que es más apegada al modelo computacional.

El tercer capítulo describe la implementación del simulador de sistemas dialógicos sobre el modelo de Lorenz, así como diversas pruebas aplicadas a él. El cuarto capítulo contiene un análisis de la aplicación de los sistemas dialógicos a la depuración de Bases de Conocimiento y una descripción de DETEC e INTERESE.

El quinto capítulo contiene una discusión sobre perspectivas del trabajo realizado y de posibles desarrollos a futuro y el apéndice describe la implementación de DIALOREN, el primer simulador.

I

Sistemas Dialógicos de Lorenzen, Lorenz y Stegmüller.

Los que ahora conocemos como "sistemas dialógicos" o como "dialéctica formal" o "teoría formal del debate" tienen una larga y notable historia. Quizás el antecedente más lejano de una teoría del diálogo, a partir del cual surge la idea de sistemas dialógicos, se encuentra en Platón. Diversos comentaristas han enfatizado que la noción platónica de "dialéctica", así como el hecho de que los trabajos de Platón se presenten en forma de diálogos, hacen pensar en una teoría subyacente del debate más que, en el primer caso, en una mera solución ad-hoc a un problema epistémico y, en el segundo, en una simple cuestión estilística.

Sin embargo, el primer tratamiento cuasi-formal de la teoría del diálogo lo encontramos en los Tópicos de Aristóteles, especialmente en el libro VIII, así como en las Refutaciones Sofísticas. Como es bien sabido, Aristóteles distingue entre varios tipos de disputa y establece diversas especificaciones sobre los papeles de "proponente" (quien defiende una tesis) y "oponente" (quien la objeta). Más aún, en la Metafísica Aristóteles aborda uno de los principales problemas de la teoría formal del diálogo: el problema de si las consideraciones dialógicas son eliminables en favor de consideraciones puramente aléticas, y responde negativamente. Aristóteles argumenta que no

es posible probar absolutamente el principio de la no contradicción, pero que sí es posible refutar a cualquier oponente que lo niegue, razonando a partir de sus propias opiniones.

El libro VIII de los Tópicos sirvió de base a lo que los lógicos medievales llamaron "el juego de las obligaciones", que consistía en cierto tipo de disputa reglada con especificaciones precisas sobre los papeles del proponente y oponente, así como sobre el tipo de locuciones permitidas durante el debate, sobre la terminación del mismo, etcétera.

Pese a las oleadas sucesivas de aristotelismo y antiaristotelismo, y a las fuertes críticas contra la lógica medieval, la tradición de la disputa reglada continuó, de alguna manera, durante el renacimiento. Llegó a perderse casi completamente entre el siglo XVIII y principios del XX. Sin embargo, en la actualidad, diversos teóricos como Lorenzen, Lorenz, Stegmüler, Rescher y Hamblin (22,19,34,33,14) han vuelto a poner de manifiesto la importancia de los "sistemas dialógicos".

En general, un sistema dialógico, tal como actualmente se entiende, contiene, esencialmente, los siguientes elementos:

- Un vocabulario, es decir, el lenguaje básico del diálogo.

- Una especificación de las locuciones permisibles a lo largo del diálogo.

- Protocolo de intercambio, que establece quién de los participantes inicia el diálogo y cuáles son las locuciones que puede emitir un participante en un paso dado del diálogo, esto es, un conjunto de reglas de preguntas y respuestas.
- Un conjunto de reglas sobre la manera de acumular o eliminar los "compromisos" de ambos participantes, esto es, las proposiciones que a lo largo de la disputa ellos se comprometen a aceptar.

Algunos sistemas, pero no todos, incluyen también reglas sobre cómo debe llevarse a cabo la terminación de la disputa, cómo se distribuye la "carga de la prueba" etcétera.

Existen diferentes formas de definir un sistema dialógico. En particular, P. Lorenzen (1971), K. Lorenz (1986) y W. Stegmüller (1964) desarrollaron una serie de sistemas dialógicos cuya intencion original era la de proporcionar una semántica para la lógica proposicional intuicionista, reemplazando la definición veritativo funcional de los conectivos por un análisis de su función dentro de ciertos juegos del lenguaje. Intuitivamente, dichos juegos son discusiones regladas entre un proponente y un oponente, donde el primero de ellos debe introducir una fórmula y defenderla exitosamente frente a los ataques del oponente; así, una fórmula será lógicamente válida si el proponente tiene una estrategia ganadora frente a cualquier oponente posible.

El sistema de Lorenzen se define por medio de dos conjuntos de reglas; primeramente, las reglas que especifican las locuciones

permisibles, la dinámica de las memorias de compromisos de los participantes y el protocolo de intercambio para cada uno de los conectivos proposicionales; en segundo lugar, define las reglas de terminación del diálogo.

Las reglas que establece Lorenzen para cada uno de los conectivos son las siguientes:

P(^): F | A ; F | B =====> F | A ^ B

O(^): F(A ^ B) , A | C =====> F(A ^ B) | C
F(A ^ B) , B | C =====> F(A ^ B) | C

P(v): F | A =====> F | A v B
F | B =====> F | A v B

O(v): F(A v B) , A | C ; F(A v B) , B | C =====> F(A v B) | C

P(-->): F , A | B =====> F | A --> B

O(-->): F(A --> B) | A ; F(A --> B) , B | C =====>F(A --> B)|C

P(~): F , A | ! =====> F | ~A

O(~): F(~A) | A =====> F(~A) | C

Donde:

A,B,C = Fórmulas proposicionales.

F = Conjunto de fórmulas proposicionales.

F(A) = Conjunto de fórmulas en el que aparece la fórmula A.

Las letras al margen izquierdo (O y P) junto con los conectivos que aparecen dentro del paréntesis adyacente, indican a qué participante en el diálogo (Oponente o Proponente) y a qué conectivo corresponde cada una de las reglas.

El símbolo "|" separa las locuciones del oponente (izquierda) y el proponente (derecha). El símbolo ";" separa las premisas de cada regla y el símbolo "," separa las fórmulas de una premisa.

Del lado izquierdo del símbolo "=====>" aparece el conjunto de condiciones que deben ocurrir dentro del diálogo para que pueda ser obtenida la que aparece a la derecha.
(i.e. (premisas) =====> (conclusión)).

De esta manera quedan determinadas las condiciones necesarias para poder demostrar una fórmula correspondiente a cada uno de los conectivos de la lógica proposicional.

Respecto a la terminación del diálogo se tienen las siguientes reglas:

F(c) ; c El proponente afirma algo que afirmó el oponente con lo cual gana el diálogo.

F(!) ; B El oponente llega a una contradicción; por lo tanto, el proponente es el vencedor.

F ! # El proponente afirma algo que no puede ser puesto en duda, por lo cual gana el diálogo.

! c El proponente no puede justificar una fórmula atómica y por lo tanto, pierde.

Donde:

! = Falsedad.

= Tautología.

El diálogo se inicia con una fórmula afirmada por el proponente (digamos a --> b) la cual se desglosa aplicando la regla (de proponente) correspondiente al conectivo principal de esa fórmula (P -->). Esta fórmula determina nuevos compromisos para cada participante, es decir, el oponente afirma a (y la guarda en su memoria de compromisos), el proponente afirma b y procede a justificarla aplicando, nuevamente, la regla (de proponente) correspondiente al conectivo principal de b.

Cabe notar que el proponente conserva la carga de la prueba excepto cuando afirma una negación, en cuyo caso la carga de la prueba pasa al oponente quien tendrá que afirmar la fórmula (positiva) que el proponente había negado y justificar alguna fórmula de su memoria de compromisos . Esta justificación la hace el oponente aplicando la regla (de oponente) correspondiente al conectivo principal de esa fórmula, estas "reglas de oponente" generan compromisos tanto para el oponente como para el proponente y revierten inmediatamente la "carga de la prueba" al proponente. El proceso continúa hasta llegar a alguna de las condiciones de

terminación.

Debe hacerse notar también que los conectivos "^" y "v" producen "subdiálogos" . En el caso del proponente, los subdiálogos generados por el conectivo "^" deben ambos ser terminados, mientras que para los generados por el conectivo "v" con que en uno de ellos se triunfe, termina el diálogo. Curiosamente, esto último sucede exactamente al revés en el caso del oponente, en donde éste debe ser capaz de ganar ambos subdiálogos del conectivo "v" y sólo uno de los del conectivo "^". Esto es porque la semántica del diálogo está orientada hacia el proponente. Por ejemplo: cuando éste impugna una fórmula con conectivo "v" (se aplica la regla del oponente) debe poder refutar ambos subdiálogos derivados para tener éxito en la demostración de la fórmula original.

Cabe aclarar que en este sistema de Lorenzen la memoria de compromisos del proponente está formada por (solamente) una fórmula, última que el proponente haya aceptado, mientras que en la memoria de compromisos del oponente se acumulan todos las fórmulas que éste haya aceptado.

Según Lorenzen, mediante las reglas anteriores es posible decidir para cada fórmula de la lógica proposicional si ella es o no un teorema de la lógica intuicionista. Por ejemplo:

Para decidir si la fórmula a --> b ==> ~b --> ~a es un teorema en la lógica intuicionista, se desarrolla el siguiente diálogo:

(0)		(a --> b) --> (~b --> ~a)	
(1)	a --> b	~b --> ~a	
(2)	~b	~a	
(3)	a	a	?(1)
(4)	! : b	: b	?(2)

En el cuarto paso, el simbolo : indica una subdivisión del diálogo en dos subdiálogos y los números entre paréntesis del lado derecho indican qué afirmación del oponente es la que el proponente está impugnando.

Tanto Lorenz como Stegmüller desarrollaron reformulaciones del sistema de Lorenzen para las cuales es posible establecer resultados de completitud. En particular, las reglas que presenta Lorenz para describir su sistema son de dos tipos. El primero, que veremos en detalle en la siguiente sección, son las reglas correspondientes a las formas permisibles de ataque y defensa para cada una de los conectivos y el segundo las reglas que se refieren a la estructura del diálogo, y que son las siguientes:

(D.1) Un diálogo consiste en una serie de argumentos presentados alternativamente por proponente y oponente.

(D.2) Cada argumento es un ataque a argumentos anteriores del contrincante o una defensa, pero no ambos a la vez.

(D.3) Los ataques se pueden hacer en cualquier momento del diálogo.

(D.4) Las defensas deben formularse en correspondencia con los ataques del adversario (sin perjuicio de aplicar la regla D.3) de tal modo que siempre debe defenderse primero la última proposición que ha sido atacada.

(D.5) Cualquier argumento puede ser atacado a lo sumo n veces por el oponente y a lo sumo m veces por el proponente.

(D.6) Las proposiciones primas (fórmulas atómicas) no se pueden atacar; el oponente las puede usar sin restricciones y el proponente puede usar sólamente proposiciones primas afirmadas antes por el oponente, y a lo sumo una vez cada una de ellas.

(D.7) Quien no es capaz de presentar un nuevo argumento pierde el diálogo.

Las reglas de Lorenz imponen restricciones al uso de fórmulas atómicas (D.6) y al orden de los ataques y defensas durante el diálogo (D.4) que no aparecen, al menos explícitamente, en el

sistema de Lorenzen. Estas restricciones son necesarias desde el punto de vista de Lorenz, para garantizar la completitud de su sistema con respecto a la lógica intuicionista (por ejemplo, señala Lorenz, si no se tiene (D.6) entonces el sistema no cumple la propiedad de sustitución (19)).

Por su parte Stegmüller desarrolló un sistema dialógico similar y presentó una demostración de su completud respecto a la lógica intuicionista. Las reglas que propone Stegmüller son las siguientes:

1.- Una regla lógica, que delimita los diferentes modos de ataque y defensa para cada conectiva y que coincide con las correspondientes al sistema de Lorenz.

2.- Una regla básica que establece la restricción de que el proponente sólo puede recurrir a la fórmula atómica si ésta ha sido previamente afirmada por el oponente.

3.- Una regla estructural, la cual prescribe lo siguiente:

a) El oponente puede atacar una sola vez, en una ronda k, una fórmula que haya sido afirmada por el proponente en una ronda anterior i. (Una ronda es una sucesión de jugadas que se inicia o se "abre" con un ataque y se "cierra" con la defensa correspondiente).

b) El proponente puede atacar un número arbitrario de veces

una fórmula afirmada por el oponente en una ronda i, siempre que cada ataque se realice en una ronda posterior a i.

c) Para defender una fórmula que aparece en una ronda i, es necesario que todas las rondas posteriores hayan sido cerradas.

Es importante destacar que si se modifica la regla estructural permitiendo al oponente un número arbitrario de ataques, entonces, el sistema resultante corresponderá a la lógica clásica.

Notemos que el sistema de Stegmüler difiere del sistema de Lorenz en las restricciones que se imponen al proponente para el uso de fórmulas atómicas, en este sentido podría considerarse el sistema de Stegmüler como un caso particular del de Lorenz y, como permite usar sólo una vez cada una de las fórmulas atómicas afirmada por el oponente, su implementación computacional es más factible.

La implementación del simulador de diálogos: DIALOG, que se hizo con base en el sistema de Stegmüller, requiere de un mayor grado de formalización, por ello, presento en el siguiente capítulo una reelaboración de las ideas de Stegmüller, la cual me permitió pasar directamente a su implementación.

II

Formalización del sistema de Stegmüller.

Definición 1: El vocabulario es el conjunto:

$$V = L_\bullet \cup \{ ?i, ?d, ? \} \cup \{ ?F : F \in L_\bullet \}$$

donde $L_\bullet$ es el conjunto de las fórmulas de la lógica de enunciados (proposicional).

Sea Σ el conjunto de todas las sucesiones contables de elementos de V, es decir:

$$\Sigma = \bigcup_{\substack{(\beta \in \omega \circ \\ \beta = \omega)}} {}^{\beta}V$$

De este conjunto de sucesiones de fórmulas debemos seleccionar aquellas que son permitidas en el juego. Para ello, es necesario establecer, primero, las reglas del juego. Esto se hará en detalle más adelante pero, por lo pronto, supondremos que las reglas del juego están resumidas en una relación binaria R sobre Σ (i.e. $R \subseteq \Sigma \times \Sigma$).

A partir de la relación R podemos determinar qué subconjunto de Σ es útil, definiendo por recursión el conjunto M de todas las jugadas permisibles en un juego dialógico para la lógica de

enunciados como sigue:

Definición 2:

$$N_0 = Le.$$

$$N_{n+1} = (\sigma \in \Sigma \ / \ \exists \gamma \in N_n, \ R(\gamma,\sigma)).$$

$$M = N_\omega = \bigcup_{(\beta \in \omega)} N_\beta$$

Llamaremos posiciones a los elementos de M y a las N_n niveles. Una posición está en algún nivel n y es, intuitivamente, la sucesión de todas las fórmulas que intervienen en un diálogo hasta un determinado momento.

Definición 3: Un sistema dialógico es una estructura de la forma $\langle M,R,w,b \rangle$ donde R es la relación que determina las reglas del juego, M es el conjunto de todas las posiciones definido a partir de R como en la definición 2 y w y b son los jugadores.

Definición 4: Si es posible, de acuerdo a R, pasar de una posición p_1 a una posición p_2, se dirá que p_2 es sucesor de p_1. El conjunto de los sucesores de una posición p se denotará: suc(p).

Cada elemento de M corresponde, intuitivamente, a un diálogo parcial, así, la relación R determina las condiciones bajo las cuales es posible pasar de un diálogo parcial a otro, esto es, las condicioones bajo las cuales uno de los jugadores puede efectuar una determinada "jugada"

Definición 5: Un diálogo es una posición R-maximal o una posición infinita.

Cada uno de los diálogos representa un debate entre los 2 jugadores acerca de la validez de una fórmula. En el debate, w propone que una fórmula es válida y b trata de refutarlo. Asi, un diálogo formal es una sucesión de elementos de V cuya primera fórmula representa la fórmula propuesta por w, la segunda fórmula es la respuesta de b y asi sucesivamente el resto de las fórmulas en la sucesión representan, alternadamente, las locuciones de cada jugador. De esta forma en una posición P∈M, ($(P_i : i \text{ es par})$) es el conjunto de las jugadas de w y ($(P_i : i \text{ es impar})$) es el conjunto de las jugadas de b.

El diálogo termina cuando, de acuerdo con las reglas del juego R, uno de los jugadores ya no puede continuar: ha quedado acorralado. En este caso diremos que el jugador que no puede continuar ha perdido el juego (diálogo) y su oponente ha ganado. Puede ser también que el diálogo no termine, en este caso se considera que w no logra probar en un tiempo finito la validez de la fórmula en cuestión, por lo tanto, pierde.

Teniendo estas definiciones es posible representar todos los posibles diálogos sobre una fórmula F en un árbol ,que llamaremos árbol de dialogos de F, de la siguiente forma: La raiz del árbol contiene la fórmula inicial F de w, $(F)\in N_o$, los siguientes nodos

contienen todos los elementos de suc((F))(subconjunto de N_i) y asi sucesivamente cada nivel contiene los sucesores del nivel anterior (fig 1).

En esta representación, cada nodo corresponde a una posición y un diálogo será,o bien una hoja del árbol o una rama infinita.

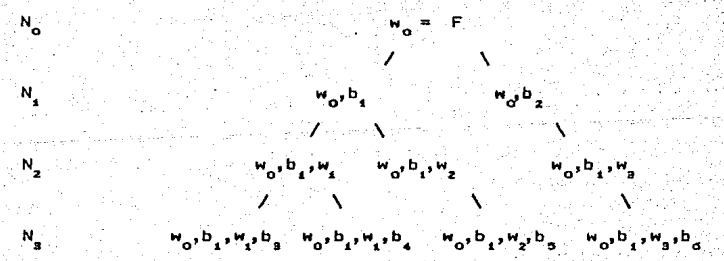


FIG 1.

Definición 6: Un diálogo g es ganado por w si y sólo si $g \in N_i$ e i es par. Correspondientemente, un diálogo g es ganado por b si y sólo si $g \in N_i$ e i es impar o si g es infinito.

De acuerdo al modelo de Lorenz, una fórmula F es d-válida (dialógicamente válida) si y sólo si w tiene una estrategia general para ganar los diálogos de F, es decir, si w ganará todo diálogo de F sin importar qué opciones tome b. Esta idea es vertida formalmente en las siguientes definiciones.

Definición 7: Una posición $p \in N_i$ con i par (posición de w) en un árbol de diálogos es favorble si y sólo si p es diálogo o $\forall s \in N_{i+1}(s \in suc(p) \rightarrow \exists t \in N_{i+2}(t \in suc(s) \wedge t$ es favorable)).

Definición 8: Una fórmula F en Le es d-válida si y sólo si, F es favorable en su árbol de diálogos.

Dependiendo de la forma en que se construya R, las definiciones anteriores permiten generar una serie de sistemas dialógicos diferentes. De hecho, R (que determina cómo pasar de una posición a otra agregando elementos de V) puede ser definida formalmente en cada caso, pero la manera más natural de entender dicha relación es en términos de un conjunto de reglas de ataque y defensa para los participantes en un diálogo (oponente y prponente, respectivamente).

En particular, para el sistema de Stegmüller, R está dada por

un conjunto de reglas lógicas (que definen los "ataques" y "defensas" permisibles para cada conectivo), básicas (que conciernen a las fórmulas atómicas) y estructurales (que establecen el número de veces y las condiciones bajo las cuales puede ser atacada o defendida una fórmula en el curso de un diálogo). Dichas reglas son las siguientes.

Reglas lógicas.

Los elementos de V están clasificados de acuerdo a su función en el diálogo como ataques y defensas. Los elementos de V cuyo primer caracter es "?", son ataques; el resto son defensas. Todas las jugadas dentro del diálogo son de alguno de estos dos tipos a excepción de la primera que es una proposición. Cada elemento de V ataca o defiende un elemento de Le que pondremos entre paréntesis para que sea claro cuál es la fórmula involucrada.

Los ataques y defensas permitidos para cada conectivo están definidos por las siguientes tablas:

Conectivo	Ataque	Defensa
¬ A	?A	
A & B	?i	A
	?d	B
	(cualquiera)	
A V B	?	A
		B

(cualquiera)

A --> B ?A B

Reglas básicas.

Sean $S_i \in \Sigma$ ($0 \leq i \leq \beta < \omega$) ; $p,q \in V$; $\phi \in Le$.

S_i,p denota la sucesión finita S_i con el elemento p al final.

$< S_i;S_i,p(\phi) > \in R$ si:

1.- p es ataque, ϕ no es atómica.

(No se pueden atacar fórmulas atómicas).

2.- β es impar, ϕ es atómica, $\phi = S_k$ para algún $k \in [0,i]$, k impar

(w solo puede afirmar fórmulas atómicas que ya hayan sido afirmadas por b).

Reglas estructurales.

Para definir las reglas estructurales, es necesario introducir antes la noción de "ronda".

Definición 9. Una "ronda" cerrada es una sucesión finita ρ_i cuyo primer elemento es un ataque y su último elemento es la defensa correspondiente. Una "ronda" abierta es una sucesión ρ_i cuyo primer elemento es un ataque y la defensa correspondiente no

aparece.

Con las mismas convenciones utilizadas para las reglas lógicas definamos las reglas estructurales:

$< S_i;S_i,p(\phi) > \in R$ si:

1.- β es par, p es ataque, $\phi = S_k$ para algún $k \in [0,i]$, k par y $p(\phi) \in S_i$.
(b puede atacar una fórmula propuesta por w una sóla vez).

2.- β es impar, p es ataque y $\phi = S_k$ para algún $k \in [0,i]$, k impar.
(w puede atacar una fórmula propuesta por b sin restricción en el número de veces).

3.- Una ronda abierta puede ser cerrada siempre y cuando contenga sólo rondas cerradas.

Modificando las reglas estructurales es posible obtener diferentes versiones del sistema dialógico de W. Stegmüller, por ejemplo, versiones correspondientes a la lógica clásica así como a varias lógicas no clásicas.

III.

Implementación del simulador DIALOG.

El programa DIALOG es un programa que simula el sistema de W. Stegmüller, esto es, dada un fórmula de la lógica proposicional, DIALOG genera todos los diálogos correspondientes a ella de acuerdo con el sistema de Stegmüller.

La implementación de DIALOG se hizo siguiendo el modelo descrito en la sección anterior, así, en el programa se hace la distinción entre los tres tipos de reglas (estructurales, lógicas y básicas) que definen la relación R del modelo teórico. Las reglas de programa (cláusulas de Horn) correspondientes a las reglas estructurales son las que guían la resolución para la generación de un diálogo, apoyándose en los otros tipos de reglas cuando sean aplicables.

El funcionamiento del programa puede resumirse como sigue: El usuario proporciona el nodo raíz del árbol, indicando de esta manera la fórmula que debe ser probada. El sistema procede entonces a generar recursivamente una rama del árbol hasta llegar a una posición terminal (diálogo completo). El sistema mantiene las locuciones de cada participante en dos listas separadas. Al final de la ejecución despliega las dos listas intercaladas en forma vertical de modo que el diálogo sea legible. La solución a la meta inicial representa un diálogo para la fórmula

proporcionada, así, al pedir todas las soluciones, se obtendrá el árbol completo correspondiente a la fórmula en cuestión (generación "en profundidad" (depth first)).

El algoritmo anterior resulta computacionalmente muy costoso como procedimiento para decidir acerca de la validez de la fórmula propuesta ya que para ello, de acuerdo a la definición 8 en la sección I, es necesario generar el árbol de diálogos casi completo. Por esto adopté la heurística de considerar el primer diálogo como el decisivo en la validez de la fórmula, que, como veremos más adelante en las pruebas al sistema, funciona adecuadamente.

Como ejemplo, en la figura 2 están los resultados proporcionados por DIALOG para las fórmulas p V - p , p --> - - p , p & (p --> q) --> q , - -p --> p.

Ahora bien, existen ciertas divergencias entre el sistema de W. Stegmüller y el sistema DIALOG; en particular, para evitar la generación de diálogos infinitos y de diálogos con repeticiones innecesarias de rondas, en el sistema DIALOG el proponente puede atacar sólo una vez cada fórmula afirmada por el oponente; esta restricción impide que algunos de los resultados obtenidos por DIALOG coincidan con los correspondientes al sistema de Stegmüller. Por ejemplo, los resultados obtenidos por el sistema de Stegmüller y por DIALOG, respectivamente, para la fórmula -- (p V - p) no coinciden entre sí, pues el siguiente diálogo, construído de acuerdo con las reglas de Stegmüller, no puede obtenerse mediante

```
> <- dialogo ( p : ^p ).

? .(p : ^ p) .nil
p .
? .^ (p) .nil
proved: dialogo(p : ^ p).
PC-Prolog:
>
>
>
>
: <- dialogo ( p --> ^(^p) ).

p .
? .(p --> ^ ^ p) .nil
^ p .
? .^ (^ p) .nil

pierde
proved: dialogo(p --> ^ ^ p).
PC-Prolog:
>
>
>
>

> <-dialogo ( ( p & (p --> q) ) --> q).

p & (p --> q) .
? .(p & (p --> q) --> q) .nil
p
p --> q

q
pierde
proved: dialogo(p & (p --> q) --> q).
PC-Prolog:
>
>
>
>
> <-dialogo ( (^(^p) ) --> p ).

^ ^ p .
? .(^ ^ p --> p) .nil

p .
? .^ (p) .nil
proved: dialogo(^ ^ p --> p).
PC-Prolog:
>
```

```
p : ^ p_ .
^ p_ .

pierde_ .

p --> ^ ^ p_ .

^ ^ p_ .

p .
? .^ (p) .nil_ .
_ .

p & (p --> q) --> q_ .

? .(p & (p --> q)) .nil_ .
? .((p --> q) & p) .nil_ .
p .
? .(p --> q) .nil_ .
q_ .
_ .

^ ^ p --> p_ .

^ p .
? .(#1 --> #2) .nil_ .

pierde_ .
```

FIGURA 2

DIALOG ya que el proponente debe atacar dos veces la fórmula - (p V - p).

Oponente	Proponente
	-- (p V - p)
-(p V - p)	p V - p
?	- p
p	p V - p
?	p

Hay argumentos que permiten asignar, teóricamente, cierta confiabilidad a los resultados obtenidos por DIALOG; a saber:

1) DIALOG puede demostrar todos los axiomas de Heyting para la lógica intuicionista (a la cual corresponde a su vez el sistema de Lorenz). Esto significa que, bajo el supuesto de que DIALOG preserva las reglas de sustitucion y modus ponens, el programa es "correcto" con respecto al sistema de Heyting y, por ende, respecto al de Stegmüller. Los axiomas de Heyting son los siguientes:

(1) p --> (p&p)

(2) (p&q) --> (q&p)

(3) (p --> q) --> ((p&r) --> (q&r))

(4) ((p --> q) & (q --> r)) --> (p-->r)

(5) q --> (p --> q)

(6) (p & (p --> q)) --> q

(7) p --> (p v q)

(8) (p v q) --> (q v p)

(9) ((p-->r) & (q-->r)) --> ((p v q) --> r)

(10) - p --> (p -->q)

(11) ((p-->q) & (p --> -q)) --> -p

En las corridas de programa adjuntas puede verse la demostración automática de cada una de las proposiciones anteriores por medio de DIALOG.

2) DIALOG es consistente, en el sentido de que no toda fórmula del sistema de W.Stegmüller es demostrable con el programa. Así, en la figura 2 se muestra que P v -P no es demostrable con DIALOG.

Desde luego, los resultados anteriores no constituyen una prueba rigurosa (en el sentido lógico del término); sin embargo, pueden considerarse suficientes para proponer la utilización del programa con bases de conocimiento específicas. A este respecto, vale la pena hacer la consideración adicional de que, en general, la corrección y eficacia en sus aplicaciones específicas, no presupone la validez general del mismo, puesto que las reglas de una base de conocimiento se conforman mediante una serie bastante limitada de esquemas (condicionales con ciertas restricciones formales sobre sus antecedentes y consecuentes).

Como parte del proceso de depuración, se ha realizado una prueba del programa DIALOG utilizando una serie de casos seleccionados como especialmente difíciles para los demostradores

de teoremas existentes a nivel lógico proposicional (30). Dichos casos son los siguientes.

(1) (p-->q) <--> (-q --> -p)

(2) --p <--> p

(3) - (p --> q) --> (q --> p)

(4) (-p --> q) <--> (-q --> p)

(5) ((p v q) --> (p v r)) --> (p v (q --> r))

(6) (p v -p)

(7) (p v ---p)

(8) ((p-->q) --> p) --> p

(9) [(p v q) & (-p v q) & (p v -q)] --> - (p v -q)

(10) --- p <--> -p

(11) [(p <--> q) <--> r] <--> [p <--> (q <--> r)]

(12) [p v (q & r)] <--> [(p v q) & (p v r)]

(13) (p <--> q) <--> ((q v -p) & (-q v p))

(14) (p <--> q) <--> ((q v -p) & (-q v p))

(15) (p --> q) v (q --> p)

(16) ((p & (q -->r)) --> s) <--> ((-p v q v s) &
(-p v -s v s)

También realicé algunas pruebas de complejidad del programa DIALOG; así, fué posible determinar que, al incrementar n en la siguiente serie de fórmulas:

U1: p1 <--> p1

U2: (p1 <--> (p2 <--> (p1 <--> p2)))

U3: (p1 <--> (p2 <--> (p3 <--> (p1 <--> (p2 <--> p3)))))

.

.

Un: (p1 <--> (p2 <--> (p3 <--> ... <-->(pn <--> (p1 <--> p2 ... <--> pn)....)

El tiempo requerido por DIALOG para generar la demostración correspondiente crece de acuerdo a la siguiente tabla.

Fórmula	Tiempo requrido (ms)
U1	89
U2	298
U3	692
U4	1322
U5	2248
U6	3520
U7	5198
U8	7358
U9	Saturación.

Estas pruebas de tiempo permiten obtener una visión del comportamiento del simulador cuando la complejidad de la fórmula a demostrar crece y de cuál es el límite en complejidad que el programa puede manejar. Es importante notar que el tiempo que aparece en la tabla representan el tiempo que tarda el programa en producir un diálogo (heurística), es por esta razón que el crecimiento no es exponencial como debía esperarse de un demostrador de teoremas para la lógica proposicional.

En suma, DIALOG es un simulador del sistema de Stegmüller que difiere de este sólo en la condición que arriba hemos mencionado y que, en términos generales, permite simular adecuadamente los diálogos formales correspondientes a dicho sistema.

IV.

Aplicaciones.

Una base de conocimiento vista como una teoría lógica puede ser inconsistente. Una base de conocimiento con tales características carecería de poder expresivo ya que si una teoría lógica T es inconsistente se tiene $T \vdash \phi$ para toda fórmula ϕ en el lenguaje de T, es decir, a partir de la base de conocimiento podría deducirse cualquier fórmula, y esto implica que la base de conocimiento no tiene utilidad.

Ante este problema han sido implementados en los sistemas comerciales procedimientos para detectar aquellos casos cuando un parámetro y su negación han sido conjuntamente instanciados o bien cuando un parámetro no es multivaluado y ha sido instanciado con más de un valor durante la consulta. Al detectar algunos de estos casos el sistema indica con un mensaje que la consulta tiene algún problema. Sin embargo, es frecuente que sistemas basados en conocimiento tengan sistemas complejos de control de inferencia que pudieran impedir que las situaciones que disparan los detectores de inconsistencias mencionados antes, no sean alcanzadas en la misma búsqueda y por tanto el usuario nunca se percate de la existencia de alguna contradicción.

Como ejemplo podemos poner una sección de una típica base de animales. Supongamos que intercaladas entre muchas otras reglas tenemos las siguientes:

R_i:si Habitat es agua entonces Animal es pez.

R_j: si Sonido es croa entonces Animal es rana.

R_k: si Sonido es croa entonces Color es verde.

Supongamos ahora que un bloque de control (escrito en pseudocódigo en español) es el siguiente:

Control:

C_1.-Escribe: "Hola, este es un experto en animales".

C_2.-Lee_datos (....,Habitat,.....).

C_3.-Establece_meta(parámetro:Animal).

C_4.-Encadena reglas hacia adelante.

C_5.-Establece_meta(parámetro:Color).

C_6.-Encadena reglas hacia atrás.

C_7.-Escribe: "El animal es" (Animal).

C_8.-Escribe: "El color del animal es" (Color).

Con la información anterior podríamos tener una consulta como la siguiente:

Hola, este es un experto en animales.

Donde vive el animal ?

-agua.

El animal buscado croa ?

-si.

Conclusión:

El animal es pez.

El color del animal es verde.

Este ejemplo muestra una base de conocimiento que junto con los datos de las consulta anterior es inconsistente. Es claro que la conclusión es incorrecta ya que el valor de Animal debió haber sido rana.

Dada la estrategia de búsqueda anterior, el sistema instancia el parámetro meta en C_4 de acuerdo a R y los datos con el valor pez. En C_5 se establece una nueva meta y en C_6 se inicia un encadenamiento hacia atrás para determinar el color del animal. De acuerdo a la regla 3, el sistema pregunta si el animal croa para verificar la hipótesis de que el animal es verde. En el segundo recorrido de las reglas (hacia atrás), el sistema no instancia la segunda regla porque la meta ya no es el parámetro Animal, entonces la contradicción no aparece y los sistemas comunes de detección no la explicitan.

Es claro que la conclusión de la consulta no es aceptada por un usuario que sepa que los peces no croan. Sin embargo si esta consulta sólo fuera un paso de una consulta más complicada que sólo despliegase las conclusiones finales (como por ejemplo, cuál, de un conjunto de lagos es el habitat natural del animal), la respuesta: "El animal vive en el lago de Patzcuaro", nos dejaría satisfechos y nos iríamos tranquilos a casa sin saber que en el lago de Patzcuaro hay peces que croan !!.

Es evidente que una contradicción como ésta es un error del creador de la base, pero este tipo de errores no son poco frecuentes, sobre todo en casos como los siguientes:

1.- El proceso de crear una base de conocimiento con el poder deductivo suficiente para ser aplicable a problemas concretos, es un trabajo largo y requiere de varias etapas de inclusión y refinamiento de reglas. En este proceso las reglas de la base

aumentan, disminuyen, se vuelven más especificas, generales etc. y en él pueden ocurrir errores que introducen inconsistencias en la base.

2.- Un sistema basado en conocimiento autocorregible es aquel que cuenta con reglas (meta-reglas) y procedimientos que le permiten modificar su propia base de conocimiento. Estos sistemas son capaces de tomar la decisión de agregar, eliminar o modificar sus propias reglas. Es difícil contemplar en la programación de dichos sistemas todas las posibles variantes de reglas que pueden ser agregadas a la base, asi, cuando el sistema agrega una nueva regla a su propia base de conocimiento, es factible que se produzca una contradicción y la base se vuelva inconsistente.

Por tanto, valdría la pena contar con una herramienta que, automáticamente, vigilara la presencia de contradicciones profundas en una Base de Conocimiento.

A continuación, se describe la implementación de una interfaz entre el entorno de sistemas expertos ESE (Expert System Environment) y DETEC, un prototipo de detección de inconsistencias que permite verificar si una consulta a una base de conocimiento proposicionalizable es inconsistente y, en caso de serlo, proponer modificaciones a la base para disolver la inconsistencia.

DETEC es un sistema que permite detectar y corregir automáticamente inconsistencias entre las reglas de la base de conocimiento y los datos provenientes de la consulta.

Básicamente DETEC está constituido por dos módulos:

a) Detector de Inconsistencias.

b) Corrector de inconsistencias.

El detector de inconsistencias recurre, a su vez, al programa DIALOG (descrito en el capítulo III) y utiliza el siguiente algoritmo para la detección de inconsistencias: dado el conjunto Ei,...,En de elementos de la base de conocimiento (reglas y datos), $E_1,...,E_n$ es inconsistente si la siguiente fórmula puede probarse como teorema:

$$E_1 \& \ldots \& E_{n-1} \text{ ---> } \neg E_n$$

El procedimiento anterior se aplica iterativamente a los datos y reglas utilizados durante una consulta; asi, si el conjunto de reglas aplicadas a la consulta es $R_1,...,R_k$ y el conjunto de datos es $D_1,...,D_m$ entonces DETEC procederá siguiendo esta secuencia:

$$\text{(I) } R_1 \text{ ---> } \neg R_2$$

si la fórmula anterior no es un teorema, el procedimiento continúa con la siguiente:

$$\text{(II) } R_1 \& R_2 \text{ ---> } \neg R_3$$

y asi sucesivamente hasta hallar una fórmula demostrable, si este no es el caso, entonces, DETEC repite el procedimiento con cada uno de los datos del conjunto $D_1,\ldots,D_m$ y partiendo de la fórmula:

$$(III)\ R_1 \& \ldots \& R_{k-1} \dashrightarrow \neg R_k$$

Asi, el conjunto de reglas de la consulta será inconsistente con el dato Di si la siguiente fórmula es un teorema:

$$(V)\ R_1 \& \ldots \& R_k \dashrightarrow \neg D_1$$

El proceso anterior se aplica igualmente a los datos $D_2,\ldots,D_m$.

En el momento en que DETEC localiza una inconsistencia entre las reglas y los datos de la base de conocimiento, entra en funcionamiento el corrector de inconsistencias, el cual genera modificaciones a las reglas involucradas en la inconsistencia y propone modificaciones al usuario el cual decide, en última instacia, cuál de las correcciones debe ser incorporada a la base de conocimiento. Asi, el corrector de inconsistencias consta, a su vez de un generador de correcciones y de una interfaz con el usuario.

El generador de correcciones recurre a la técnica de añadir premisas a las reglas involucradas en la inconsistencia y el algoritmo general que utiliza es el siguiente:

i) Elige dos reglas con conclusiones contradictorias, del conjunto de reglas.

ii) Selecciona de una de ellas, las premisas que no aparecen en la otra.

iii) Genera una corrección agragando la negación de una de las premisas seleccionadas a la regla en donde ésta no aparece. Este proceso se llama restricción de la regla.

iv) Verifica, mediante el detector de inconsistencias, cuáles de las correcciones generadas eliminan efectivamente la inconsistencia.

Para cada premisa seleccionada existe una posible corrección. Una vez localizadas las correcciones que eliminan la inconsistencia, DETEC propone al usuario la lista de ellas para que él elija la mas conveniente.

Para propósitos de este trabajo entenderemos una base de conocimiento proposicionalizable como una B.C. cuyos parámetros son booleanos,bivaluados o numéricos.

Los problemas que presenta la implementación de una interfaz entre DETEC y ESE son varios. Por un lado, es necesario aislar las reglas y los datos que intervienen en la consulta cuya

consistencia queremos verificar.Por el otro, dado que el lenguaje en el que se representa el conocimiento en el entorno ESE es un especie de pseudo lenguaje natural, es necesario convertir esta información en fórmulas proposicionales para poder ser procesadas por el detector de inconsistencias DETEC.

El primer problema se resuelve sustrayendo la información del archivo que contiene el seguimiento completo de la consulta (este archivo lo genera ESE después de cada consulta). El segundo problema que, ciertamente es el más complicado, se resuelve traduciendo el texto de cada regla utilizada y los datos de la consulta a fórmulas proposicionales por medio del siguiente procedimiento:

Como la base de conocimiento debe ser proposicionalizable, los parámetros pueden haber sido definidos como booleanos directamente o se les pudo haber asignado un dominio con 2 valores.

En el primer caso los parámetros serán traducidos como letras proposicionales con el mismo nombre si su valor es TRUE y como la negación del nombre si su valor es FALSE.

Por ejemplo: supongamos que tenemos un parámetro llamado "TRABAJO" definido como booleano. Su traducción como variable proposicional será TRABAJO si el valor del parámetro es TRUE y ¬TRABAJO si su valor es FALSE.

En el caso en que el parámetro tiene un dominio con 2 valores el sistema lee del archivo de la base de conocimiento el primer valor del dominio y lo considera como valor "esperado". De esta

manera la traduccion de un parámetro con dominio de 2 valores será:

-El nombre del parámetro, guión y el valor esperado (PARAMETRO_VALOR), si su valor coincide con el valor esparado.

-La negación del nombre del parámetro, guión, el valor esperado (¬ PARAMETRO_VALOR) si su valor no coincide con el valor esperado.

Esta es la forma en la que son traducidos los datos de la consulta, (la instanciación de los parámetros) a letras proposicionales.

Las reglas se traducen cambiando los conectivos verbales (y,o,si-entonces y no) por sus símbolos correspondientes (&, |, --> y ¬).

Las relaciones se traducen a letras proposicionales ligando con un guión el nombre de la relación (a veces escrito de otra manera) y sus argumentos instanciados. Se introduce el símbolo ¬ cuando la relación es verificable y no se cumple de acuerdo a la instanciación de los predicados. Ejemplos:

La relación: edad <= 35

se traduce como: EDAD_MENOR_IGUAL_35 si el valor del parámetro edad es menor o igual que 35

y como: EDAD_MAYOR_35 si el valor del parámetro edad no es menor

o igual que 35.

La relación: animal es rana

se traduce como: ANIMAL_RANA si el valor asignado al parámetro animal es rana

y como ¬ ANIMAL_RANA si el valor asignado al parámetro animal no es rana.

El procedimiento completo sigue la siguiente secuencia:

1.-Después de una consulta ESE genera un archivo que contiene el desarrollo completo de ésta.

2.-La interfaz DETEC-ESE, al ser activada, extrae el valor esperado de todos los parámetros de la base de conocimiento y del archivo que contiene la consulta.

3.-En seguida, aisla los valores asignados a cada uno de estos parámetros durante la interacción con ESE.

4.-Posteriormente lee el texto de las reglas involucradas y los envía al traductor.

El traductor utiliza la gramática de ESE, no para decidir si una regla es aceptable por el sistema (como suelen usarse las gramáticas), sino para generar un árbol sintáctico en el que aparecen clasificados todos los componentes de la regla de acuerdo a su función.

En un archivo separado está almacenada la gramática de ESE en forma DCG (Definite Clause Grammar) (35) esto permite accesarla

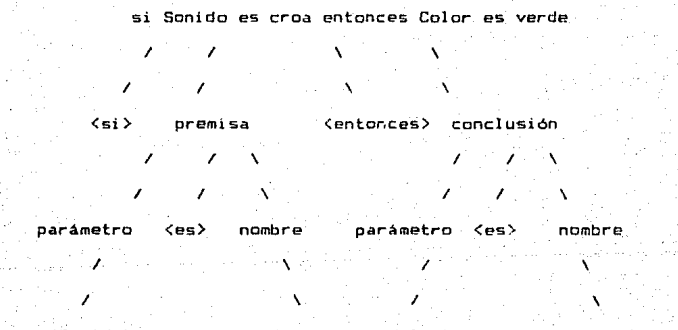
con cualquier editor de textos y que su lectura sea fácil. Para que esta gramática sea usada por el traductor escrito en Prolog, es necesario transformarla a cláusulas de Horn. Esto se hace con la ayuda de un programa escrito también en Prolog.

Un ejemplo del árbol sintáctico generado es el siguiente:

La regla:

si Sonido es croa entonces Color es verde

Se representa como:



Este árbol permite seleccionar fácilmente las palabras relevantes para la forma proposicional de la regla y, también, para, una vez corregida una regla, poder reconstruirla fácilmente e incorporarla a la base de conocimiento.

5.- Una vez obtenidos los árboles sintácticos de las reglas, un conjunto de reglas de traducción elaboran la versión lógica correspondiente a cada una de las reglas de la consulta.

6.- Estas reglas traducidas a Lógica Proposicional son incorporadas a DETEC quién verifica si, junto con los datos, son inconsistentes.

7.- En caso de hallar inconsistente el conjunto de reglas y datos, el sistema presenta un conjunto de modificaciones a las reglas de la consulta, generado por DETEC para resolver el problema y deja al usuario la elección de una corrección.

8.- Una vez que el usuario eligió la modificación más conveniente, el sistema retraduce la versión modificada de la regla con la ayuda del árbol sintáctico para finalmente actualizar la base de conocimiento original con la corrección.

Este sistama fué programado en VMPROLOG y corre en una IBM 4381. Hasta ahora lo hemos probado con bases prototipo para ver que funciona ya que el probador de teoremas que utiliza DETEC agota la pila (stack) de la máquina rapidamente (con pocas reglas).

Un ejemplo de la ejecución de INTERESE es el siguiente:

La base de conocimiento sobre recomendaciones de crédito es la siguiente: (La mezcla de idiomas se debe a la sintaxis de ESE)

R_1. if trabajo is no and casa is no then recomendacion es denegada.

R_2. if trabajo is si and casa is si and crédito>5 m. and crédito<20 m. and familires<3 then recomendación is autorizado.

R_3. if trabajo is si and casa is si and crédito>5 m. and crédido<20 m. and familiares>7 then recomendación is denegada.

R_4. if trabajo is si and casa is si and familiares>3 and familiares<7 and cuenta_bancaria>10 m. then recomendación es denegada.

R_5. if trabajo is si and casa is si and crédito>20 m. and plazo is 12 meses then recomendación is denegada.

R_6. if trabajo is si and casa is si and coche is si then solvencia is si.

R_7. if solvencia is si and tarjeta is si then recomendación es autorizada.

Ejecución:

SISTEMA DE DETECCION DE INCONSISTENCIAS EN ESE

SNY..1989.

Valores esperados de los parámetros:

PARAMETRO(TRABAJO,'si').

PARAMETRO(CASA,'si').

PARAMETRO(RECOMENDACION,'autorizado').

PARAMETRO(SOLVENCIA,'si').

PARAMETRO(TARJETA,'si').

PARAMETRO(COCHE,'si').

Leyendo archivo de la ultima consulta.....

Asignaciones durante la consulta:

ASIGNACION(TRABAJO,'si').

ASIGNACION(CREDITO,10).

ASIGNACION(CASA,'si').

ASIGNACION(FAMILIARES,10).

ASIGNACION(TARJETA,'si').

ASIGNACION(COCHE,'si').

Reglas traducidas de la consulta:

R_8

R_6

R_7

Hay una contradicción entre los datos y las leyes de la consulta:

Datos:

CASA

FAMILIARES_MAYOR_8

CREDITO_MAYOR_5

CREDITO_MENOR_20

TARJETA

TRABAJO

Reglas:

SOLVENCIA & TARJETA --> AUTORIZADO.

TRABAJO & CASA & COCHE --> SOLVENCIA.

TRABAJO & CASA & CREDITO_MAYOR_5 & CREDITO_MENOR_20 & FAMILIARES_MAYOR_8 --> ¬ AUTORIZADO.

Te gustaría una corrección que dificulte o facilite el crédito ?

D.

Las correcciones que disuelven la cotradicción son las siguientes:

¬ FAMILIARES_MAYOR_8 & SOLIDEZ & TARJETA --> AUTORIZADO

¬ CREDITO_MENOR_20 & SOLVENCIA & TARJETA --> AUTORIZADO

¬ CREDITO_MAYOR_5 & SOLVENCIA & TARJETA --> AUTORIZADO

¬ CASA & SOLVENCIA & TARJETA --> AUTORIZADO

¬ TRABAJO & SOLVENCIA & TARJETA --> AUTORIZADO

Te parece apropiada esta corrección ?:

¬ CREDITO_MENOR_20 & SOLIDEZ & TARJETA --> AUTORIZADO.

NO.

Te parece apropiada esta correción ?:

¬ TRABAJO & SOLVENCIA & TARJETA --> AUTORIZADO.

SI.

La base de conocimiento ha sido corregida.

V.

DIALOREN y DIALOG en IA.

DIALOREN puede ubicarse dentro de una serie de trabajos en I.A., que han pretendido simular exitosamente diversas formas del razonamiento dialógico; me refiero a trabajos como el de Doyle y Collins-Stevens (8). En términos generales el interés por los diálogos en I.A. encuentra una de sus motivaciones en la posibilidad de lograr realizaciones en el campo del aprendizaje automático y del aprendizaje asistido por computadora (4).

Como se ha mencionado en la sección anterior, DIALOG es un simulador del sistema de Stegmüller, y a partir de él se podría obtener un demostrador de teoremas para la lógica proposicional intuicionista si además de generar todos los diálogos posibles para una fórmula dada, DIALOG tuviese un mecanismo para determinar si dicho conjunto de diálogos indica o no que la fórmula es un teorema.

Hasta ahora, no tenemos un algoritmo que permita a DIALOG tomar automáticamente la decisión, pero podemos afirmar que aún utilizando criterios bastante rudimentarios, DIALOG funciona de manera bastante eficiente como demostrador de teoremas. Por ejemplo, si nuestro criterio consiste en tomar en cuenta sólo el primero de los diálogos generados, obtenemos un resultado satisfactorio en las pruebas descritas en la sección III.

El "demostrador de teoremas" que resulta, al añadir a DIALOG el método de decisión que he mencionado, se ha utilizado, con resultados exitosos, para la detección de inconsistencias dentro de una base de conocimiento. Sin embargo, la confiabilidad de DIALOG para aplicaciones como la mencionada dependerá de contar con un procedimiento de decisión más sofisticado lo cual es, como he mencionado, un problema abierto.

Ambos simuladores de diálogos pueden tener diversas aplicaciones:

(a). En la medida en que DIALOREN y DIALOG simulan adecuadamente sistemas dialógicos como los de Lorenzen y Lorenz, se prueba que es posible manejar computacionalmente, de modo adecuado, nociones dialógicas de importancia central como: "oponente","proponente" y "carga de la prueba". Este resultado permite a su vez, aproximarse a la modelación de situaciones dialógicas más complejas, como las involucradas en los distintos temas de la deliberación y el aprendizaje.

(b). De modo más inmediato, DIALOREN y DIALOG podrían utilizarse en el campo del aprendizaje de la lógica intuicionista como medio para probar automáticamente teoremas de esta lógica y compararlos con los de la lógica clásica. Este resultado no es poco importante, en términos prácticos, si se considera que es fácil construir un verificador de teoremas para la lógica clásica basado en tablas de verdad, pero para el caso de la lógica intuicionista no existe un método directo que permita verificar la validez de las fórmulas.

c) En el campo conocido como inteligencia artificial distribuida (IAD), en el que la interacción entre agentes es de vital importancia, los sistemas dialógicos podrían ser una herramienta útil como protocolo de comunicación, construyendo un sistema dialógico con más de 2 participantes, o como medio para resolver conflictos de consistencia entre agentes. En esta aplicación sería muy importante que el sistema dialógico utilizado funcionara para la lógica modal ya que se ha propuesto la utilización de esta lógica en los sistemas de IAD como paradigma para representar las relaciones entre agentes (7).

d) La elaboración de reglas para sistemas expertos involucra el enfrentamiento con diversos problemas de consistencia como el descrito en la sección de aplicaciónes (IV). Otro de estos problemas surge durante la ingeniería del conocimiento, cuando las reglas son generadas por inducción, a partir de ejemplos, tomados de varios expertos (personas). Las reglas generadas tendrán distintos grados de especificidad y representarán las opiniones de los expertos sobre el mismo tema. Esta diversidad de opiniones puede generar una base de conocimiento inconsistente por lo que sería útil contar con una herramienta de depuración de bases de conocimiento en construcción que, a medida que se agreguen reglas, verifique que cada regla agregada no haga inconsistente el conjunto. Para esta labor podrían utilizarse técnicas de detección como las descritas en la sección IV pero sería útil, usar un sistema dialógico para un subconjunto apropiado de la lógica de primer orden con dominio finito, en lugar de uno proposicional como el descrito en dicha sección. Esto es debido a que en el

ejemplo de la sección IV se supone una base en la que los parámetros son bivaluados y un sistema como el que propongo ahora necesita ser más general y poder manejar cuantificadores para poder tomar en cuenta todas las instanciaciones de las fórmulas para poder detectar adecuadamente las inconsistencias que pudierar surgir.

Bibliografía.

1 - Abbagnano N. et al., La evolución de la Dialéctica., Martinez Roca, S.A.,España, 1981.

2 - Angelleli, I. "The Technics of Disputation in the History of Logic". Journal of Philisophy, 67,(1970).

3 - Aristóteles, Organon, Editorial de ciencias sociales, La Habana Cuba, 1975.

4 - Barr A, Feigenbaum E.(eds) The Handbook of Artificial Intelligence, V.2.,William Kaufmann Inc. Los Altos California, 1982 pp.242-246.

5 - Barwise, Keisler, Kunen, eds. The Kleene Symposium, Proceedings of the Symposium held June 18-24, 1978 at Madison, Wisconsin, U.S.A.

6 - Bundy A. "An Analytical Comparison of Some Rule-Learning Programs" Artificial Intelligence 27,(1985), pp.137-181.

7 - Castillo L. "On Distributed Artificial Intelligence", Tesis de maestría, Universidad de Edimburgo, Junio, 1988.

8 - Doyle J. "A model for deliberation, action and introspection", Tech. Rep.; AI-TR-581, MIT,(1980).

9 - Enderton H.B. Una introducción matemática a la Lógica. UNAM, 1987.

10 - Expert System Consultation Environment and Expert System Development Reference Manual (SH20-9609 -2).

11 - González, Negrete, Noriega. "Un diseño para bases de conocimiento evolutivas". Memorias de la V Reunión Nacional de Inteligencia Artificial. Limusa, México, 1988 pp. 221-32.

12 - González E. Noriega P. "Aproximaciones a un modelo dialógico de bases de datos", Memorias de la III Reunión Nacional de Inteligencia Artificial, Oaxaca, México, 1986.

13 - González E., Noriega P. "Los sistemas dialógicos y algunas de sus aplicaciones", Memorias de la II Reunión Nacional de Inteligencia Artificial, Monterey N.L., México, 1985.

14 - Hamblin, Ch. Fallacies, Methuen, Londres, 1970.

15 - Hayes-Roth "Using Proofs and Refutations to Learn from Experience" en Michalski, Carbonell y Mitchell, Machine Learning.

An Artificial Intelligence Approach, Tioga, Palo Alto, Cal, 1983, pp 221-40.

16 - Hodges, Wilfred. Building Models by Games, Cambridge University Press, 1985.

17 - Kowalski, R. Logic for Problem Solving. North-Holland, 1979.

18 - Lorenz, K. "Rules v.s. Theorems", Journal of Philosophical Logic, 2, pp 352-369, (1973).

19- Lorenz, K. "Rules vs Theorems. A New Approach for the Mediation Between Intuitionistic and Two-Valued Logic" Journal of Philosophical Logic 2 pp. 191-216 (1986).

20 - Lorenzen, P. Normative Logic and Ethics, Zurich, 1967.

21 - Lorenzen, P. Pensamiento Metódico, Sur, Buenos Aires, 1969.

22 - Lorenzen, P. Metamatemática, Tecnos, Madrid, 1971.

23 - Mackenzie, J.D. "The Dialectics of Logic", Logique et Analyse #5.

24 - Negrete S. "INTERESE: Un detector de inconsistencias para el entorno ESE (Expert System Environment)", Memorias de la V reunión nacional de Inteligencia Artificial, Oro, México, 1989.

25 - Negrete S., Noriega P. y González E. "DIALOREN: Un simulador de diálogos formales" Memorias de la 1a. reunión iberoamericana de Inteligencia artificial (IBERAMIA), Barcelona, 1988.

26 - Negrete S. y González E. DETEC: Guía del usuario. Reporte de investigación, Centro Científico IBM de México, Mexico, 1989.

27 - Negrete S., González E. y Noriega P. "Sistemas dialógicos y demostración automática de teoremas" Memorias de la V reunión nacional de Inteligencia artificial (VIRNIA), Oro. México 1989.

28 - Nguyen T., Perkins W., Laffey T. y Pecora D. "Knowledge Base Verification" Artificial Intelligence Magazine (1987) pp. 69-75.

29 - Noriega P. y González E. "En torno de un modelo dialógico de base de datos", Comunidad Informática, no. 26, México 1986.

30 - Noriega, P y González, E. "Retórica, Dialéctica y cambio de creencias. Un nuevo punto de vista sobre la justificación de las teorías del debate.". Estudios No.5, (1985).

31 - Pelletier (1986) "Seventy-Five Problems for Testing Automatic Theorem Provers" Journal of Automated Reasoning 2 (1986) pp. 191-216.

32 - Pereira F., Shieber S. Prolog and Natural Language Analysis CSLI, Stanford CA, 1987.

33 - Rescher, N. Dialectics, New York State University Press, Albany, 1977.

34 - Stegmüller, Wolfgang."Remarks on the completness of logical systems relative to the validity-concepts of P. Lorenzen and K.Lorenz ",Notre Dame Journal of Formal Logic, Vol. V, Number 2, April 1964.

35 - Sterling L. Shapiro E. The art of PROLOG. MIT Press.

36 - Valls Plana R. La Dialéctica.Montesinos, Barcelona, 1981.

37 - VM/Programming in Logic, Program Description/Operations Manual., IBM, 1985. 5785-ABH.

38 - Walker A., Mc.Cord M., Sowa J. y Wilson W. Knowledge Systems and Prolog. Addison-Wesley,1987.

Implementación de DIALOREN.

En DIALOREN, el simulador correspondiente al sistema de Lorenzen, se han descrito las reglas de Lorenzen para los participantes en el diálogo en forma de cláusulas de Horn como es usual en la programación en Prolog.

Las reglas describen de una manera determinista (34) cómo deben hacer frente cada una de las partes a la impugnación o defensa de una fórmula, de acuerdo con la forma general de ésta. Por ejemplo: una de las cláusulas define qué debe responder el oponente cuando el proponente le ha pedido la justificación de una fórmula que en su forma general es un condicional.

La elección de la regla aplicable a este caso la hace el sistema unificando la forma de la fórmula, dada en notación prefija (ejemplo: cond(a,b) se lee "si a entonces b") con la cabeza de la cláusula que tiene como argumento al condicional. Este proceso es una característica intrínseca del lenguaje Prolog.

En la versión actual del programa, la memoria de compromisos de las locuciones del oponente ha sido implementada como una estructura de datos de tipo "cola" (queue) de modo que, al estar el proponente en posición de impugnar alguna de las afirmaciones

anteriores del oponente, escogerá la más antigua del almacén.

El diálogo se inicia invocando al proponente a través del predicado dialogo dándole la fórmula a discutir como único argumento. A continuación, el sistema decidirá si el diálogo ha terminado, o bien, cuál o cuáles son las subfórmulas que deberán discutirse para poder verificar la anterior. Esto se lleva a cabo llamando recursivamente al predicado diálogo o al predicado oponente, dependiendo de quién de las partes tiene la carga de la prueba. Al identificar la regla aplicable al tipo del conectivo general de la fórmula en cuestión, siguiendo el proceso antes referido, el sistema abre nuevos subdiálogos y éstos a su vez derivan en otros hasta que se llegue a alguna de las condiciones de término que a continuación se citan.

El diálogo (o subdiálogo) puede llegar a su término de dos maneras:

(1). Se origina un diálogo de la forma dialogo(F) en donde F es atómica y la máquina detecta que F se halla en la "cola" (compromisos del oponente). En este caso el predicado que representa al diálogo en pugna adquiere el valor de verdadero.

(2).Sucede una situación análoga a la anterior pero F no esta en la cola. En este caso el predicado adquiere el valor de falso.

Semánticamente, el valor de verdad de los predicados está visto desde el lado del proponente; es decir, si los predicados dialogo y oponente adquieren el valor de verdadero, significará

que el proponente ha tenido éxito en su intento por "convencer" al oponente de que la fórmula propuesta, es una tautología.

Cada vez que un subdiálogo llega a su fin, adquiere un valor de verdad. En este momento, regresa a resolver el subdiálogo inmediato superior (backtracking) y así sucesivamente hasta retornar el valor final del diálogo. Mientras el programa recorre el árbol de posibilidades, las locuciones de ambas partes son impresas como efectos laterales (side-efects) a través de predicados como "write". En caso de que algún subdiálogo haya terminado, el sistema imprime un aviso indicando quién ha ganado; entonces continúa.

La programación del sistema en Prolog no sólo presenta ventajas sobre otros lenguajes, sino que parece más natural dado que la formalización del diálogo por Lorenzen se basa en un conjunto de reglas lo que encaja perfectamente con la filosofía de Prolog. Así mismo, el uso del recorrido hacia atrás en el árbol de posibilidades (backtracking) para volver a los subdiálogos superiores, ahorra un gran esfuerzo de programación y añade legibilidad, naturalidad y belleza al programa. Finalmente, el hacho de asignar un valor de verdad a cada diálogo o subdiálogo, representados como predicados dentro del programa, define una semántica apropiada al problema que se trata.

Limitaciones y Extensiones.

Cuando en el transcurso del diálogo, aparecen distintas posibilidades de demostración a partir de un estado de la memoria de compromisos, DIALOREN actualmente selecciona el compromiso mas antiguo como aquél sobre el que procede el diálogo. Esta regla de selección, aunque correcta, no necesariamente es la única posible. Lorenzen no hace explícita ninguna.

DIALOREN, en su última versión, genera distintos diálogos ante algunos conectivos como "^" dependiendo del orden de sus argumetos (conyuntos). Esto significa que el diálogo producido, aunque posiblemente exitoso, no muestra una estrategia general de ganancia contra cualquier posible oponente, como pretendía Lorenzen. e.g. (a v b) ^ ~b ---> a. En la figura 3 está el diálogo generado por DIALOREN para esta fórmula.

Lorenzen incluye, además de las reglas representadas en DIALOREN, dos pares de reglas más que definen el manejo de cuantificadores. La versión próxima de DIALOREN deberá extenderse para contener toda la lógica intuicionista de primer orden.

```
<- dialogo( ((A ; B) & ¬B ) -> A).
GOAL : <- DIALOGO((A ; B) & ¬B -> A) .
'opo: '(A ; B) & ¬B .
'pro ----------------------------------> ''impugno '(A ; B) & ¬B .
'pro ----------------------------------> ''1 impugno 'A ; B .
'opo: '' pendiente ' ¬B .
'opo: 'A ; B .
'opo: 'A .
'opo: ''supongo 'A .
'pro ----------------------------------> ''2 impugno '¬B .
'opo: '¬B .
'pro ----------------------------------> 'B'?' .
'pro ----------------------------------> ''no se puede saber si 'B .
'pro ----------------------------------> 'A .
'  gana -------> el proponente afirma algo que afirmó el oponente
10MS SUCCESS
 <- DIALOGO((A ; B) & ¬B -> A) .
                                                   VM READ   CCMEXVMS
```

FIGURA 3

Listados anexos.

Simulación de diálogos sobre los axiomas de Heyting para la lógica intuicionista.

```
> ?- dialogo ( p --> (p&p) ).

p .
? .(p --> p & p_ .nil
? .(p & p_ .nil
pierde
proved: dialogo ( p --> p & p).
PC-Prolog:
>
>
>
> ?- dialogo ( p & q ) --> ( q & p ) ).

p & q .
? .(p & q --> q & p) .nil
? .(q & p) .nil
p
q
? .(p & q) .nil
pierde
proved: dialogo ( p & q --> q & p).
PC-Prolog:
>
>
>
>
> ?-dialogo ( p --> q ) --> ( (p&r) --> (q&r) ) .

p --> q .
? .(p --> q --> (p & r --> q & r) .nil
p & r .
? .(p & r --> q & r) .nil
? .(q & r) .nil
p

q
r
? .(r & q) .nil
pierde
proved: dialogo ( p --> q --> p & r --> q & r).
PC-Prolog:
>
>
>
>
>
>
>
```

```
p --> p & p_ .

p & p_ .
p_ .
_ .
```

```
p & q --> q & p_ .

q & p_ .
? .(p & q --> q & p_ .nil_ .
? .(q & p --> q & p_ .nil_ .
q_ .
p_ .
_ .
```

```
p --> q --> p & r --> q & r_ .

p & r --> q & r_ .
q & r_ .
? .(p & r --> q & r_ .nil_ .
p .
? .(p --> q --> p & r --> q & r_ .nil_ .
? .(r & p --> q & r_ .nil_ .
q_ .
r_ .
_ .
```

```
>
> <- dialogo ( ( (p-->q) & (q-->r) ) --> (p-->r) ).
                                                    (p --> q) & (q --> r) --> p --> r_ .
(p --> q) & (q --> r) .
? .((p --> q) & (q --> r) --> p --> r) .p --> r_ .
p .
? .(p --> r) .nil                                   ? .((p --> q) & (q --> r)) .nil_ .
p --> q                                             p .
                                                    ? .(p --> q) .nil_ .
q                                                   ? .((q --> r) & (p --> q)) .nil_ .
q --> r                                             q .
                                                    ? .(q --> r) .nil_ .
r                                                   r_ .
pierde                                              _ .
proved: dialogo((p --> q) & (q --> r) --> p --> r).
PC-Prolog:
>
>
>
>
>
>
>
>
>

> <-dialogo( q --> ( p --> q) ).
                                                    q --> p --> q_ .
q .
? .(q --> p --> q) .nil                             p --> q_ .
p .
? .(p --> q) .nil                                   q_ .
pierde                                              _ .
proved: dialogo(q --> p --> q).
PC-Prolog:
>
>
>
>
> <- dialogo( ( p & (p-->q) ) --> q ).
                                                    p & (p --> q) --> q_ .
p & (p --> q) .
? .(p & (p --> q) --> q) .nil                       ? .(p & (p --> q)) .nil_ .
p                                                   ? .((p --> q) & p) .nil_ .
p --> q                                             p .
                                                    ? .(p --> q) .nil_ .
q                                                   q_ .
pierde                                              _ .
proved: dialogo(p & (p --> q) --> q).
PC-Prolog:
>
```

```
? <- dialogo ( p --> (p | q) ).
                                        p --> p | q_ .
p .
? .(p --> p | q) .nil                   p | q_ .
? .(p | q) .nil                         p_ .
pierde                                  _ .
proved: dialogo(p --> p | q).
PC-Prolog:
>
>
>
>
? <- dialogo ( (p | q) --> (q | p) ).
                                        p | q --> q | p_ .
p | q .
? .(p | q --> q | p) .nil               q | p_ .
? .(q | p) .nil                         ? .(p | q) .nil_ .
p                                       p_ .
pierde                                  _ .
proved: dialogo(p | q --> q | p).
PC-Prolog:
>
>
>
? <- dialogo ( ( (p-->r) & (q-->r) ) --> ( (p|q) --> r ) ).
                                        (p --> r) & (q --> r) --> p | q --> r_
(p --> r) & (q --> r) .
? .((p --> r) & (q --> r) --> p | q --> p | q --> r_ .
p | q .
? .(p | q --> r) .nil                   ? .((p --> r) & (q --> r)) .nil_ .
p --> r                                 ? .((q --> r) & (p --> r)) .nil_ .
q --> r                                 ? .(p | q) .nil_ .
p                                       p .
                                        ? .(p --> r) .nil_ .
r                                       r_ .
pierde                                  _ .
proved: dialogo((p --> r) & (q --> r) --> p | q --> r).
PC-Prolog:
>
>
>
>
>
>
? <- dialogo( ^p --> (p --> q) ).
                                        ^ p --> p --> q_ .
^ p .
? .(^ p --> p --> q) .nil               p --> q_ .
p .
? .(p --> q) .nil                       p .
                                        ? .^ (p) .nil_ .
pierde                                  _ .
proved: dialogo(^ p --> p --> q).
PC-Prolog:
>
```

```
> <- dialogo( ( (p --> q) & (p --> ^q) ) --> ^p).
(p --> q) & (p --> ^ q) --> ^ p_ .

(p --> q) & (p --> ^ q) .
? .((p --> q) & (p --> ^ q) --> ^ p) .nil_ p_ .

p .
? .^ (p) .nil
p --> q

q
p --> ^ q

^ q

pierde
proved: dialogo((p --> q) & (p --> ^ q) --> ^ p).
PC-Prolog:
>
>
>
>
:
```

```
? .((p --> q) & (p --> ^ q)) .nil_ .
p .
? .(p --> q) .nil_ .
? .((p --> ^ q) & (p --> q)) .nil_ .
p .
? .(p --> ^ q) .nil_ .
q .
? . (q) .nil_ .
_ .
```